DSA-Assignment-6

V. Sai Deepika
AP19110010415
CSE-H

1, Take the elements from the user and sort them in descending order and do the following
a, Using binary search find the element & the location is the array where the element is asked from user
b, Ask the user to enter any two locations print the sum & product of values at those location in sorted array

Sol

```
#include <stdio.h>
int main()
{
 int i, low, high, mid, n, key, array[100], temp, one, two, sum, product;
 printf ("Enter the number of elements in array");
 scanf ("%d", &n);
  printf ("Enter %d integers", n);
 for(i=0; i<n; i++)
 scanf ("%d", &arr[i]);
  for (i=0; i<n; i++)
  {
    if [i= i+1; j<n; j++)
    { if [arr[i] < arr[j])
    { if (temp= arr[j]);
    {
      arr[i]= arr[j];
      arr[j] = temp;
    }
```

```
}
}
printf("\n elements of array is sorted in descending
                                        order \n");
for (i=0; i<n; i++)
{ printf ("%d", arr[i]);
} printf ("Enter values to find");
  scanf ("%d", &key);
  low = 0
  high = n-1
  mid = (low + high)/2;
  while (low < high)
  {
    if (arr[mid] > key)
    {
      low = mid + 1;
    }
    else if (arr[mid] = key)
    { printf("%d found at location %d", key, mid+1);
    break;
  }
   else
   high = mid - 1;
   mid = (low + high)/2;
   }
  if (low > high)
  {
  printf("Not found %d isnt present in list \n", key);
 }printf ("%.n");
printf("Enter two locations to find sum & product");
scanf("%d", &one);
```

```
scanf("%d", &two);
sum=(arr[one]+arr[two]);
product=(arr[one]*arr[two]);
printf("The sum of elements=%d", sum);
printf("The product of elements=%d", product);
return 0;
}
```

output:- Enter number of elements in array 4

Enter 4 integers

2

3

1

7

elements of array is sorted in descending order

7 3 2 1

Enter value to find 3

3 found at location 2

Enter two locations to find sum & product of elements

1

7

The sum of elements = 8

The product of elements = 7

8, Sort the array using merge sort where elements are taken from the product of kth elements from first and last where k is taken from the user

Sol:-

```
# include <stdio.h>
# include <conio.h>
# define MAX_SIZE 5
void merge_sort[MAX_SIZE];
void merge_array(int, int, int, int);
```

```
int arr_sort[MAX_SIZE];
int main()
{
int i, k, pro=1;
printf("sample merge sort example functions & array\n");
printf("\n Enter %d elements for sorting", MAX_SIZE);
for(i=0; i<MAX_SIZE ;i++)
{
scanf ("%d", &arr_sort[i]);
printf ("In your data:");
{
for (i=0; i<MAX_SIZE ;i++)
{ printf ("if %d", arr_sort[i]);
}
merge_sort [0, MAX_SIZE-1];
printf ("\n sorted data:");
for (i=0; i<MAX_SIZE ; i++)
{ printf ("\t %d", arr_sort[i]);
}
printf ("find the product of k^th elelemt from i^st & last\n");
scanf ("%d", &k);
pro= arr_sort[k]* arr_sort[MAX_SIZE-1];
printf ("product = %d", pro);
get ch()
}
void merge-sort (int i, int j)
{ int m;
if (i<j)
{ m=(i+j)/2
```

```
merge_sort(i, m);
merge_sort(m+1, j);
merge_array(i, m, m+1, j);
}
}
void merge_array(int a, int b, int c, int d)
{
int t[50];
int i=a, j=c, k=0;
while(i<b && j<=d)
{
if(arr_sort[i] < arr_sort[j])
t[k++] = arr_sort[i++];
else
t[k++] = arr_sort[j++];
}
while(j<=b)
t[k++] = arr_sort[j++];
for(i=0; j=a; i<=d; i++, j++)
arr_sort[i] = t[j];
```

Output:

```
sample merge sort example - functions & array
enter 5 elements for sorting
9
7
4
6
2
your data : 9 7 4 6 2
sorted data 2 4 6 7 9
find the product of kth elements from 1st
last where k=2
product = 36.
```

Discuss insertion sort & selection sort with example

Sol Insertion Sort:- Insertion sort works by inserting the set of values in the existing sorted file. It constructs the sorted array by inserting a single element at a time. This process continues until whole array is sorted in same order. The primary concept behid insertion sort is to insert each item into its appreciate place in final list. The insertion sort method saves an effective amount of memory.

Working of insertion sort:-

* It uses two sets of arrays where one stores the sorted data and other on unsorted data.
* The sorting algorithm works until there are elements in the unsorted set.
* Let us assume there are 'n' numbers elements in array. Initially, the element with index 0 (LB=0) exists in the sorted set remaining elements are in unsorted portion of list
* The first element of the unsorted position has array index (if B=0)

Advantages of insertion sort:-

* Easily implemented and very efficient when used with small sets of data.
* The additional memory space requirement of insertion sort is less (ie, (d!))
* It is faster than other sorting algorithms.

complexity of insertion sort:-

The best case complexity of insertion sort is O(n) times, i.e; when the array is previously sorted. In the same way, when the array is sorted in reverse order, the first element in unsorted array is to be composed with each element in sorted set. so, in the worst case running time of insertion sort is quadratic, i.e($O(n^2)$). In average also it has to make the minimum $(k-1)/2$ comparisons. Hence, the average case also has quadratic running time $O(n^2)$.

Example:-

arr[] = 46 22 11 20 9

// Find the minimum element in arr[0---4] & place at begining

9 46 22 11 20

// Find the minimum element in arr[1---4] & place at begining arr[1---4]

9 11 46 22 20

// Find the minimum element in array[2---4] & place at begining

9 11 20 46 22

// Find the minimum element in arr[3---4] & insert at begining

∴ sorted array

9 11 20 22 46.

Selection Sort:- The selection sort perform sorting by searching for minimum value number and placing it into the first or last posistion according to the order. The process of searching the minimum key and placing if in the proper position is continued until all the elements are placed at right position.

Working the selection sort:-

→ Suppose an array with n elements in memory.

→ In the first pass, the smallest key is searched along with its position, then the arr[pos] is supposed & swapped with arr[0]. Therfore arr[0] is sorted.

* In the second pass, again the posistion of smallest value is determined in subarray of (n-1) elements inter charge the arr[pos] with Arr[1]

* In the pass (n-1), the same process is performed to sort the n no. of elements.

Advantages of selection sort :-

→ The main advantage of selection sort in that is performs well on a small list.

→ Further more, because is an in-place sorting algorithm, no additional temporary storage is required beyond what is needed to hold the original list

* Complexity of selecting sort:-

Thus, the running time complexity of selection sort is $O(n^2)$

$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} = O(n^2)$

Example:-

13 12 14 6 7

Let us loop for i=1 to 4

i=1. Since 12 is smaller than 13, move 13 & insert 12 before 13 do same for i=2, i=3, i=4

∴ Sorted array is 6 7 12 13 14

(4) Sort the array using bubble sort where elements are taken from the user & diplay the elements

i, In alternative order

ii, Sum of elements in odd positions & product of elements in even position

iii, elements which are divisible by m where m is taken from user

Sol:-

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int arr[50], i, j, n, temp, sum=0, product=1;
  printf("Enter the total number elements to sort");
  scanf("%d", &n);
  printf("enter %d elements:", n);
  for(i=0; i<n; i++)
  scanf("%d", &arr[i]);
  printf("\n sorting array using bubble sort technique
                                        \n");
  for(i=0; i<(n-1); i++)
  {
```

```
for (j=0; j<[illegible]
{
  if (arr[j] > arr[j+1]
  { temp = arr[j];
    arr[i] = arr[j+1];
    arr[j+1] = temp;
  }
}
printf("All array elements sorted successfully: \n");
printf("array elements in ascending order: \n\n");
for (i=0; i<n; i++)
{ printf("%d \n", arr[i]);
}printf("array elements in alternate order \n")
for (i=0; i<=n; i = i+2)
{ printf("%d\n", arr[i]);
}
for (i=1; i<=n; i=i+2)
{ sum = sum + arr[i];
}printf("The sum of odd position elements are = %d \n", sum
for (i=0; i<=n; i=i+2)
{ product *= arr[i]
}
printf("The product of even position elements are
                                %d \n", product);
getch();
return 0;
}
```

output:-

Enter total no. of elements to store = 5

Enter 5 elements

8
4
6
3
1

Sorting array using bubble sort technique

All array elements sorted successfully.

Array elements in ascending order

1
3
4
6
8

Array elements in alternate order

1
4
8

The sum of odd position element is 9

The product of even position elements are 32

(5) write a recursive program to implement binary search?

```
# include <stdio.h>
# include <stdlib.h>
void binary search (int arr[], int num, int first, int last)
{
  int mid;
  if (first > last)
  {
    printf("Number is not found"); }
  else {
    mid = (first + last / 2; }
  if (arr[mid] == num) {
```

```
printf("element is found at index %d", mid);
exit(0);
}
else if (arr[mid] > num)
{ primary search (arr, num, first, mid -1);
}
else
{
Binary search (arr, num, mid +1, last);
}
}
void main ()
{ int arr[100], beg, mid, end, i, n, num;
printf ("Enter the size of an array");
scanf ("%d", &n);
print f("enter the values in sorted sequences \n");
for (i=0; i<n; i++)
{scanf ("%d", &arr[i]);
}
beg=0;
end = n-1;
printf ("enter a value to be search:");
scanf ("%d", &num);
Binary search (arr, num, beg, end);
}
```

Output:-

```
Enter the size of an array 4
Enter the values in sorted sequence
1
2   Enter a value to search: 4
3
4   Elements is found at index: 3
```